Small Logic Gates Spawn Big Dreams — Part 1

Yes, Michael, there is such a thing as a generic logic chip.
Learn how to turn your junk box orphans into a construction project or replacement part treasures.

by TJ Byers

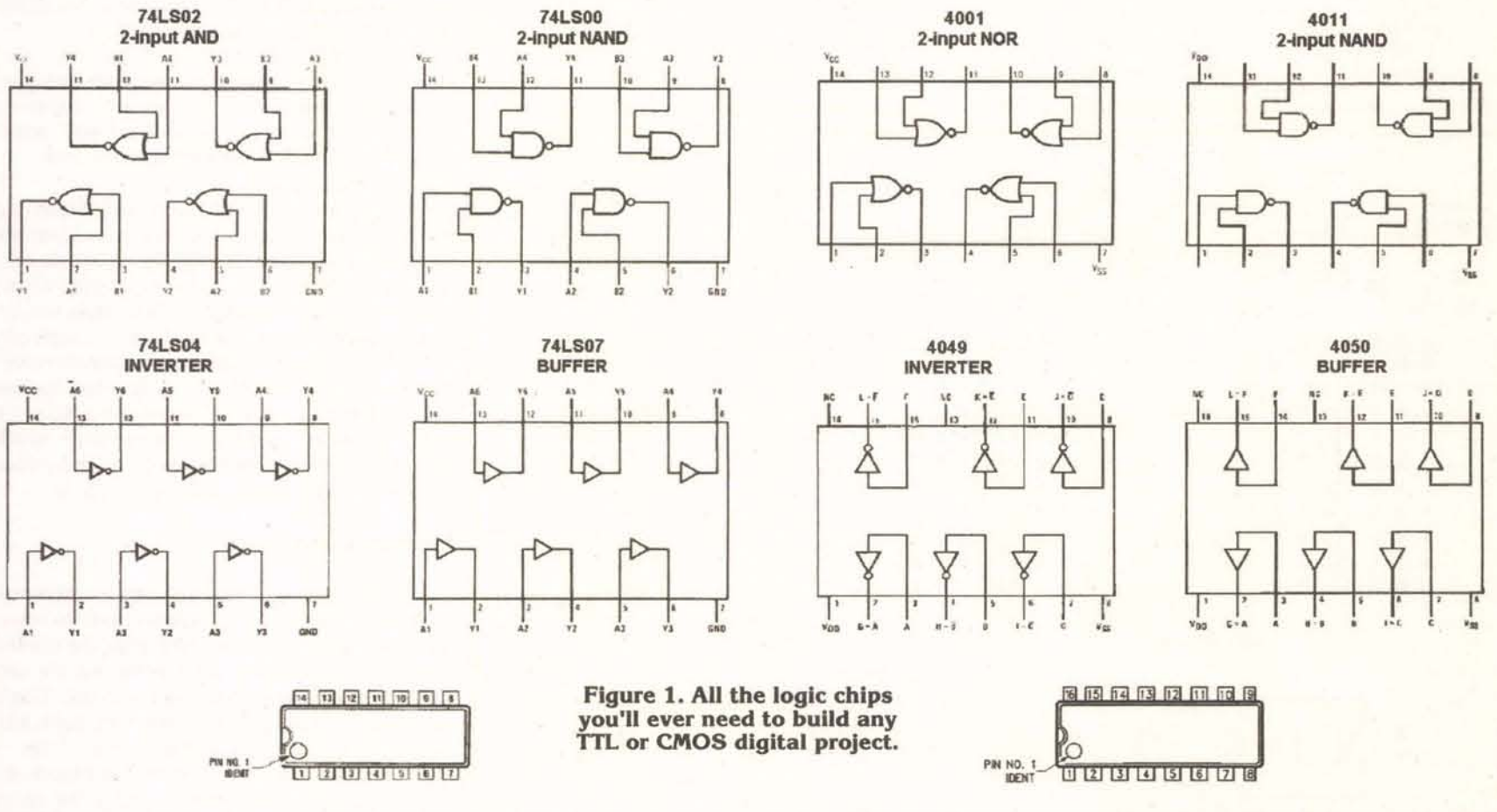


Figure 1. All the logic chips you'll ever need to build any TTL or CMOS digital project.

Straight from the "It never fails" files. Ten o'clock Saturday night. You're putting the finishing touches on your mindboggling new robot steering controller. Visions of Bot Wars games numb your senses. Just one more step, a simple RS flip-flop, and your beast will roar into action. But wait! The IC cupboard is ... well, bare. Not a flip-flop in sight!

Not to fret, though. If you have just a single NOR or NAND chip in that rat's nest, you have the makings of an RS flip-flop. In fact, you probably have more logic functions in there than you're aware of.

All logic circuits — no matter how complex (including that BASIC Stamp microcontroller) — are based on two elementary propositions: AND and OR. Throw in an inverter gate (NOT), and you can solve any logic equation.

In this article, I'll show you how to use these basic building blocks to get you out of a parts jam, whether it be for a new project or as a replacement device. You will also learn about the subtle differences between seemingly similar logic functions, so that you can choose the flavor that best suits your pet project.

The Basics

Boolean algebra is the basis for all computer logic. Originally formulated by George Boole, an English mathematician (1815-1864), Boolean algebra describes propositions whose outcomes are either true or false. In computer work, the true and false statements are represented by the output state of an electronic circuit, whose state can be either 1 (true) or 0 (false). We perceive these logic changes as voltage levels. That is, when the logic level is supposed to represent a 1, the output voltage will typically measure five volts. Conversely, a logic 0 is represented by — what else? Zero volts. This is important. Logic 1 is five volts and logic 0 is no volts. (Okay, you nitpickers. Before you pluck my pupils out, let me say that voltages other than five volts can represent a logic 1, with 3.3 volts being the newest inductee.)

The basic operators are AND, OR, and NOT. AND is the product of the two arguments; that is, if both propositions are true, then the statement is true. If either of the propositions is false, then the statement is false. OR is the sum of the propositions. If either proposition is true, or both are true, then the statement is true. NOT is the inverse or complement of a statement. If the statement is true, then its complement is false.

Expressed algebraically, the Boolean equations are:

Expression	Proposition	Statement
x • y	x AND y	true
x + y	x OR y, x AND y	true
/x • /y	NOT x AND NOT y	true
/x + /y	NOT x OR NOT y, NOT x AND NOT y	true

The inverse function is an important logic operator because it can be used to simplify otherwise complex functions, and is sometimes the

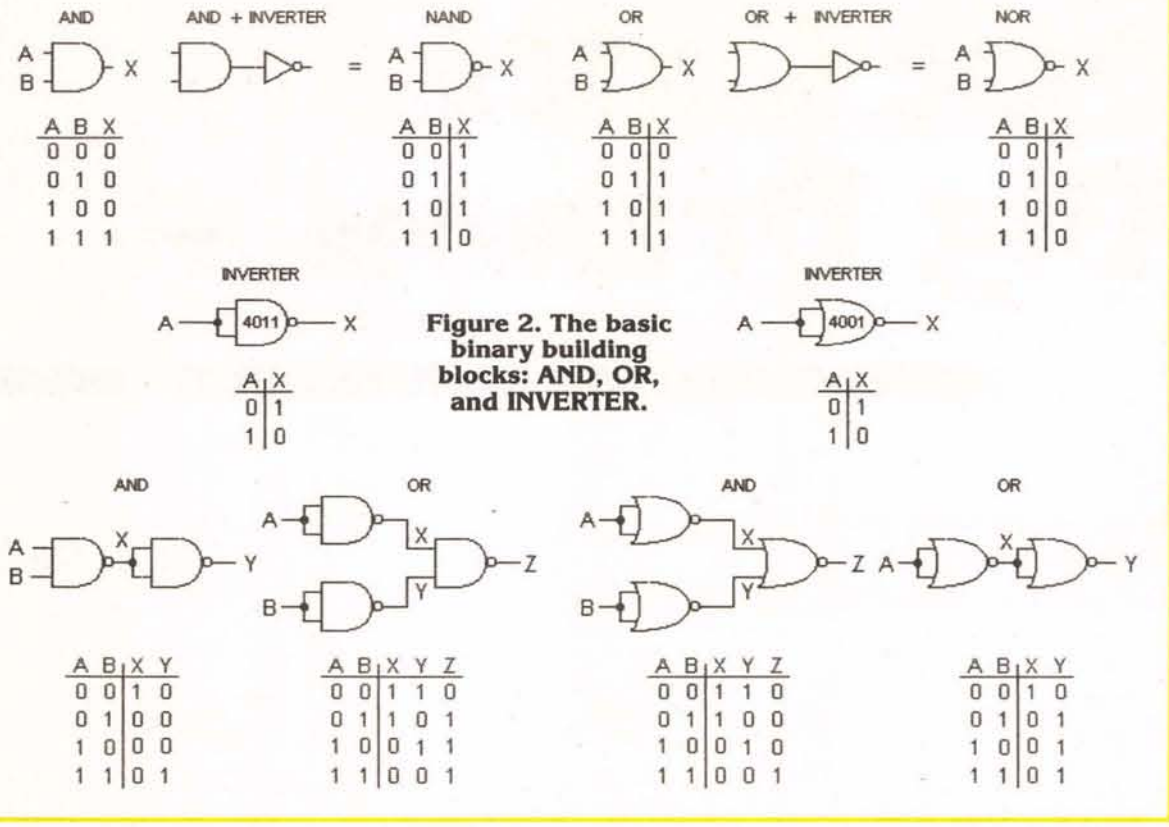


Figure 2. The basic binary building blocks: AND, OR, and INVERTER.

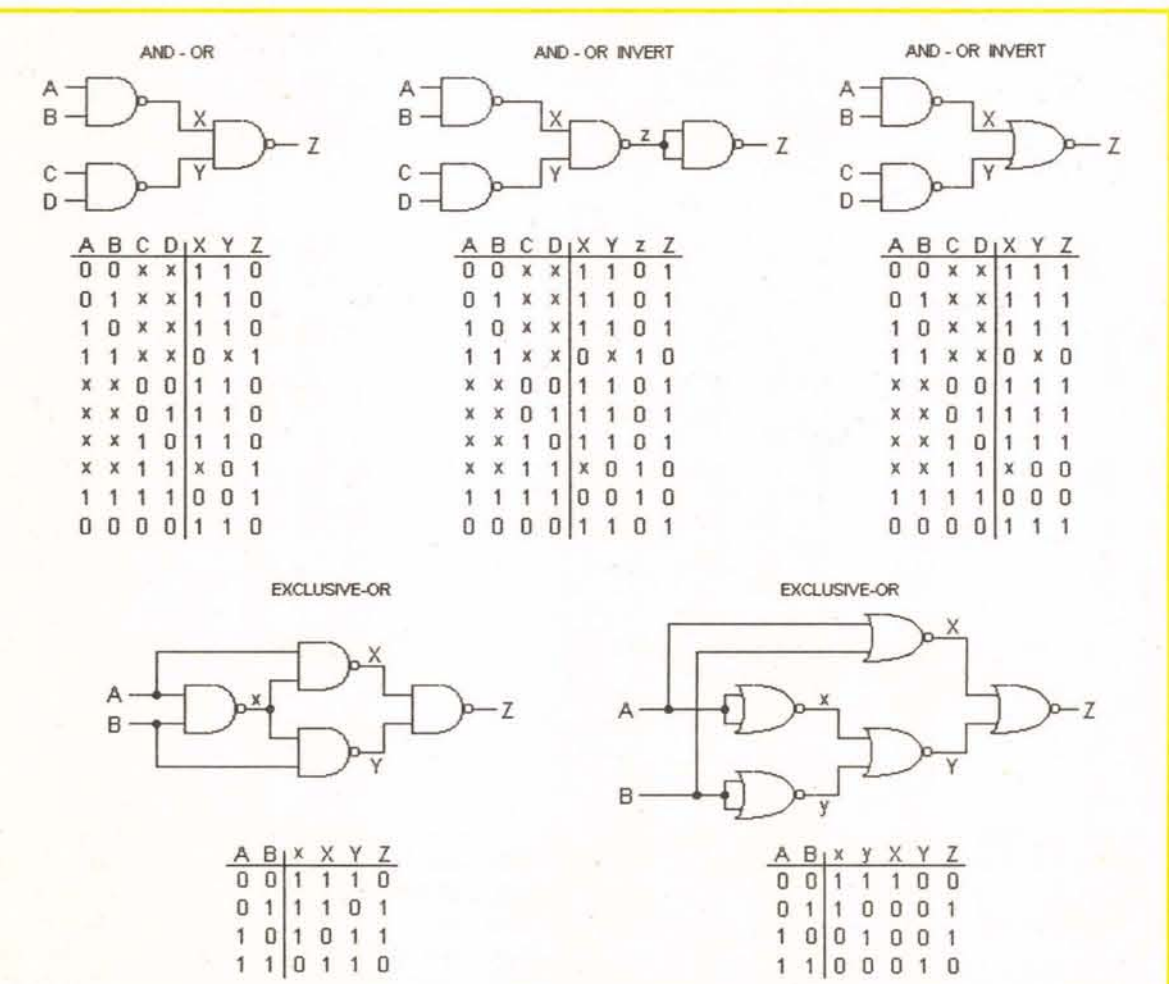


Figure 3. The AND-OR and exclusive-OR gates are a perfect example of complex logic functions, made from negative logic gates, that find widespread use in the computer industry.

only way to arrive at a solution. For example, it's virtually impossible to solve the equation of

x OR y, but not x AND y

(the logical reasoning of an exclusive OR gate) without the NOT operator. And without the exclusive-OR gate, computers wouldn't be smart enough to add or subtract. Figure 1 shows the logic chips that we'll use for the rest of this discussion, and Figure 2 shows their elementary structures.

Early logic gates were built on RTL (resistor-transistor logic) technology, which was plagued by noise, blatantly cantankerous, and downright slow. RTL construction techniques of the time also limited the gate types to the simpler AND, OR, and inverter functions. It wasn't until the advent of TTL (transistor-transistor logic) that more complex designs could be successfully fabricated.

The advances included multiple inputs beyond four propositions, stable edge triggering, and increased fan-out capability. But none were more important than the paradigm shift from positive (AND/OR) logic to negative (NAND/NOR) logic thinking (Figure 3). Negative logic simplifies many Boolean equations, which leads to a reduced gate count — which results in very sophisticated logic products like embedded microcontrollers and Pentium processors.

Negative logic isn't hard to understand, but it takes a nimble mind to implement it efficiently. In this discussion, there is a heavy emphasis on negative logic and the clustering of gate types to reduce IC count. Most logic ICs contain two or more identical gates, many of which might go unused if you don't know how to wire them to perform operations other than for what they were intended. For example, an unused NAND or NOR gate can easily substitute for an inverter, which could possibly eliminate an extra IC and make the product less costly and more reliable.

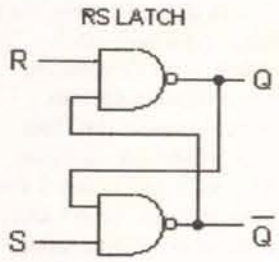
I Remember, Too

Logic gates can also be used as memory elements. In order for a logical circuit to remember and retain its logical state after the controlling input stimulus has been removed, it's necessary to include some form of feedback. This is easily done using a pair of inverters, each having its input connected to the output of the other, like the cross-linked circuit in Figure 4.

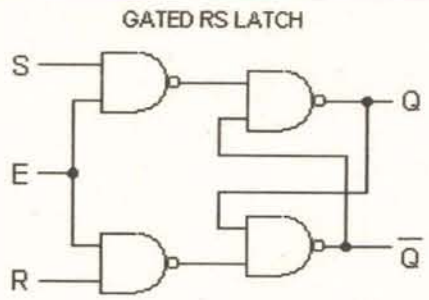
Because the gates are inverting, the outputs will always have an opposite logic level of the input. Depending on which gate assumes control on power-up, this circuit will latch up in one of two states. To toggle between these states, an extra input is added to the cross-linked inverters, like the NAND latch shown in Figure 5a.

The inputs are traditionally designated S and R for Set and Reset, respectively, and the outputs are designated Q and /Q The designations S and R are a logical evolution, but what prompted the use of Q for an output? Legend has it that the early memory gates weren't totally reliable, and so researchers labeled the predicted outcome as Questionable (so the story goes). In a commercial latch circuit, either or both outputs may be available for use by other circuits.

For the NAND latch circuit, both inputs should normally be at logic 1. Pulling the top gate's input low (logic 0) forces its output high (logic 1). This logic 1 is also applied to the input of the bottom NAND gate, forcing that output low (logic 0). This, in turn, feeds back to the second input of the original gate, thereby clos-

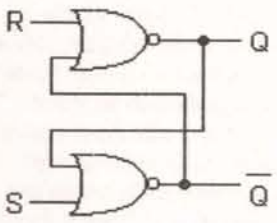


R	S	Q	$\overline{Q}$
0	0	not allowed	
0	1	1	0
1	0	0	1
1	1	no change	

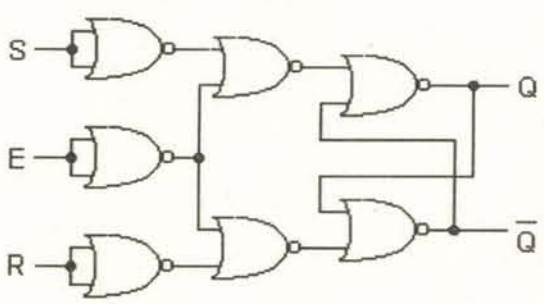


E	R	S	Q	$\overline{Q}$
1	0	0	not allowed	
1	0	1	1	0
1	1	0	0	1
1	1	1	no change	
0	x	x	no change	

Figure 6. The RS NAND latch is created by adding a pair of NAND gates to the input circuits of a basic RS latch.



R	S	Q	$\overline{Q}$
0	0	no change	
0	1	1	0
1	0	0	1
1	1	not allowed	



E	R	S	Q	$\overline{Q}$
1	0	0	no change	
1	0	1	1	0
1	1	0	0	1
1	1	1	not allowed	
0	x	x	no change	

ing the feedback loop. The gates are now locked (Figure 5b), even if the /S input signal is removed. Applying logic 0 input to the top gate will have no further effect on this circuit. However, applying a logic 0 to the bottom gate will cause the same reaction in the other direction, thus changing the state of the latch (Figure 5c). If you need to switch states on a positive note, pulling the inputs high (logic 1) instead of low (logic 0), a NOR gate may be substituted for the NAND gate.

Note that it's forbidden to have both NAND inputs at a logic 0 level at the same time (the opposite occurs in the NOR latch configuration). That state will force both outputs to a logic 1, overriding the feedback latching action. In this condition, whichever input goes to logic 1 first will lose control, while the other input (still at logic 0) controls the resulting state of the latch. If both inputs go to logic 1 simultaneously, the result is a "race to the finish line" frenzy, and the final state of the latch cannot be determined ahead of time.

Pardon My Hiccups

RS latches are the most simple-minded of the flip-flop family — but that doesn't prevent them from contributing to the digital community. RS latches are often used to eliminate contact bounce in mechanical switches. When you press the button or change a switch's position, the physical contacts flex a little, causing them to make and break several times before settling down. You don't notice this when turning on a light in your home, but digital circuits are fast enough that they do notice this behavior and mimic it faithfully.

The solution is to insert an RS latch between the mechanical switch and the following circuitry, as shown in Figure 5d. Notice that the unconnected input is held at a logic 1 via a pull-up resistor, while the switched input is held at logic 0 by a direct connection to ground. (The resistor prevents the switch from shorting Vcc to ground.) When the switch is moved to its alternate position, the gate toggles to its opposite state and latches in place; additional bounces will have no effect on the output. This eliminates the contact bounce and sends a single, clean digital pulse to the next circuit.

While RS latches are useful devices, they have their limitations. One problem is how to control when the latch is allowed to change state, and when it's not. This is necessary if we have a group of latches and want to be sure they all change state (or not) at the same time. By adding a pair of NAND gates to the input circuits of the RS latch, we accomplish two goals: normal rather than inverted inputs, and a third input common to both gates which we can use to synchronize this circuit with others of its kind.

The gated RS NAND latch (Figure 6) circuit is very similar in operation to the basic latch. The S and R inputs are normally at logic 0, and must be changed to logic 1 to switch the state of the latch. However, with the third input, a new factor has been added. This input is typically designated clock or CLK, because it's typically controlled by a clock circuit of some sort, which is used to synchronize banks of these latch circuits with each other. The output can only change state while the CLK input is a logic 1. When CLK is a logic 0, the S and R inputs have no effect.

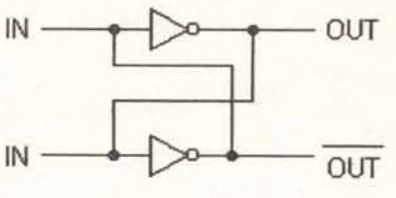


Figure 4. In order for a logical circuit to be used as a memory cell, it's necessary to include some form of feedback. By cross-linking the outputs of two inverters to its opposite inputs, the circuit forms a stable memory retention element.

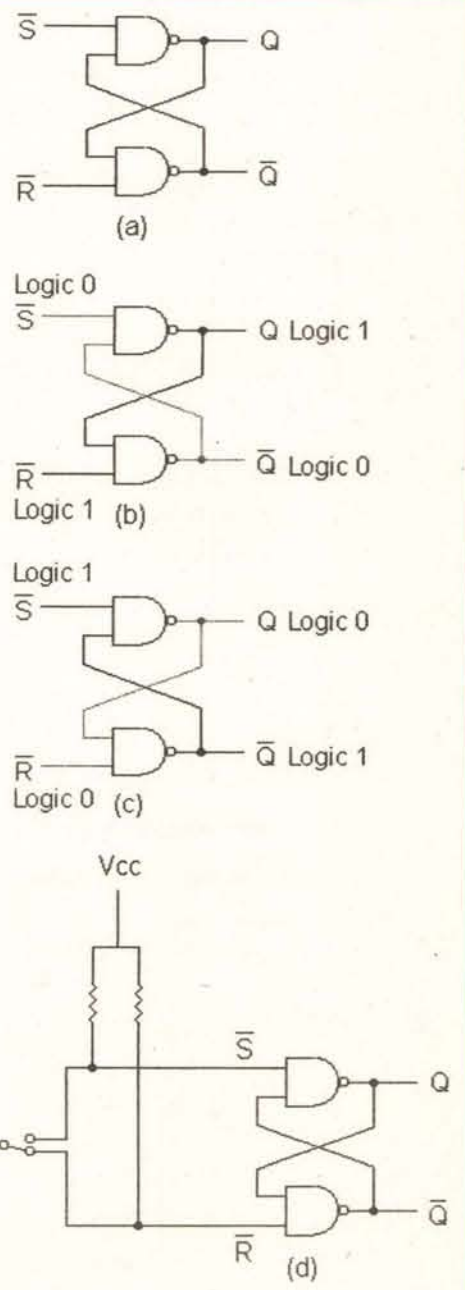


Figure 5. An RS latch can be used to debounce mechanical switches, sending a single, clean digital pulse to the next circuit.

Flip-Flops, Not Pop-Tops

Although the gated RS latch solves some of the problems of basic RS latch circuits, and

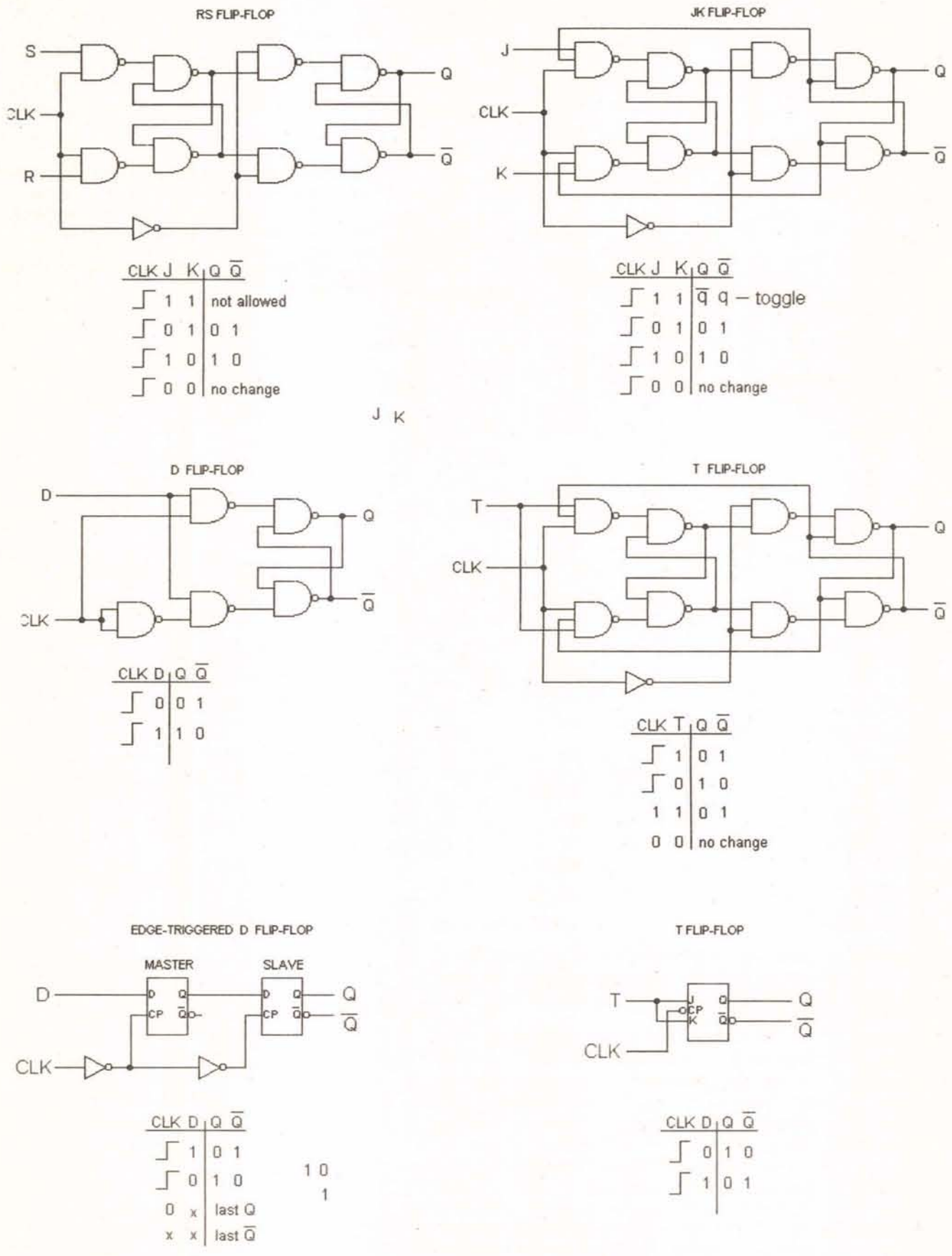


Figure 7. Flip-flops come in a wide variety of sizes, shapes, and features.

allows closer control of the latching action, it's by no means the last word. The RS latch circuit could easily experience a change in S and R input levels while the CLK input is still at a logic 1 level. This allows the circuit to change state many times before the CLK input returns to logic 0.

A good way to make sure that the latch can only change its outputs exactly on the clock cycle trigger is to sample the inputs on the transition, or edge, of the clock signal. To adjust the gated RS latch for edge triggering, we must use two identical gated latch circuits, each of which operates on opposite halves of the clock signal. The resulting circuit is commonly called a flip-flop, because its output can first flip one way and then flop back the other way. The two-section flip-flop is also known as a master-slave flip-flop, because the input latch operates as the master section, while the output section is slaved to the master during half of each clock cycle.

The edge-triggered RS NAND flip-flop is shown in Figure 7. As previously described, the circuit actually consists of two identical RS latch circuits, with an inverter connected between the two CLK inputs to ensure that the two sections are enabled during opposite half-cycles of the clock signal. The inverter is the key to the operation of this circuit.

If we start with the CLK input at logic 0, the S and R inputs are disconnected from the input (master) latch. Therefore, any changes in the input signals cannot affect the state of the final outputs. When the CLK signal goes to logic 1, the S and R inputs are able to control the state of the input latch, just as with the single RS latch. At the same time, though, the inverted CLK signal is applied to the output (slave) latch and prevents the state of the input latch from having any effect here. Therefore, any changes in the R and S input signals are tracked by the input latch while CLK is at logic 1, but are not reflected at the Q and /Q outputs.

When CLK falls again to logic 0, the S and R inputs are again isolated from the input latch. At the same time, the inverted CLK signal now allows the current state of the input latch to reach the output latch. As a result, the Q and /Q outputs can only change state when the CLK signal falls from a logic 1 to logic 0. This is known as the falling edge of the CLK signal; hence the designation "edge-triggered flip-flop."

By making the flip-flop edge-triggered, we can precisely control the moment when all flip-flops will change state. This arrangement also allows plenty of time for the master latch to respond to the input signals, and for those input signals to change and settle down following the previous change of state.

JK Flip-Flop

Still, there is one problem left to solve: the ambiguous output condition which may occur if both the S and R inputs are at logic 1 when the CLK falls from logic 1 to logic 0. In the example above, we automatically assume that the master latch will end up in the logic 1 state — but this isn't always a certainty with real components. Therefore, we need a way to prevent a race between the two gates to see which one reaches logic 1 state first.

The solution is to add extra feedback from the slave latch to the master latch. This modified latch is called a JK flip-flop. To prevent any possibility of a "race" occurring between the R and S inputs when both inputs are at logic 1, we must somehow prevent one of the inputs from having an effect on the master latch. At the same time, we still want the flip-flop to be able to change state on each falling edge of the CLK input if the input logic calls for it. Whether the S input or the R input is to be disabled depends on the current state of the slave latch outputs. If the Q output is a logic 1 (the flip-flop is in the

Set mode), the S input can't be any more set that it already is. Therefore, the S input can be disabled without disturbing the flip-flop under these conditions. In the same way, if the Q output is logic 0 (the flip-flop is Reset), the R input can be disabled without causing any harm. We have thus prevented a foot race to logic 1.

Feedback control of the RS flip-flop inputs is attained by adding two new inputs from the Q and /Q outputs. (Remember that NAND and NOR gates may have any number of inputs.) To prevent confusion, the designations of the new flip-flop inputs are designated J (instead of S) and K (instead of R).

In most ways, the JK flip-flop behaves just like the RS flip-flop. The Q and /Q outputs will only change state on the falling edge of the CLK signal, and the J and K inputs will control the future output state pretty much as before. However, there are some notable differences.

Since one of the two logic inputs is always disabled, the master latch can't change state back and forth while the CLK input is at logic 1. Instead, the enabled input can change the state of the master latch once. This isn't true of the RS flip-flop. The Q and /Q outputs can only change state with each falling edge of the CLK signal, with the master latch circuit changing state with each rising edge of CLK.

We can use this characteristic to our advantage in a number of ways. A flip-flop built specifically to operate this way is typically called a T flip-flop (Toggle flip-flop). The lone T input is, in fact, the CLK input for other types of flip-flops.

The operation of the T flip-flop is identical to that of an RS flip-flop with joined S (set) and R (reset) inputs. Every T input pulse triggers switching of the latch to the opposite state. When it's in logic 0, a CLK pulse changes the T flip-flop to the logic state 1. When the latch is in logic 1, a clock pulse changes the flip-flop to logic state 0. Note that the frequency of the output pulses is exactly half of the frequency of the input pulses.

D Flip-Flop

One very useful variation on the RS latch circuit is the Data latch, or D flip-flop, as it's generally called. The D flip-flop is constructed by using the inverted S input as the R input signal. The single remaining input is designated D to distinguish its operation from other types of latches. It makes no difference that the R input signal is effectively clocked twice, since the CLK signal will either allow the signals to pass both gates or it won't.

When the CLK input is logic 1, the Q output will always follow and reflect the logic level present at the D input, no matter what course it may take. However, when the CLK input falls to logic 0, the state of the D input at that instant is trapped and held in the latch for use by whatever other circuits may need this signal. Because the single D input is also inverted, this latch circuit can't experience an ambiguous state condition caused by all inputs being at logic 1 simultaneously, making it safe to use in any circuit.

Although the D flip-flop doesn't have to be edge-triggered, there are some applications where an edge-triggered D flip-flop is desirable. This can be accomplished by using a D latch circuit as the master section of an RS flip-flop. Both types are useful, and both are commercially available.

With all of these different types of latches and flip-flops, the logic diagrams we've been using have gotten rather large, especially for the edge-triggered flip-flops. Fortunately, it really isn't necessary to follow and understand the inner workings of any of these circuits when they are used in larger applications. Instead, we used a set of very simple symbols to represent each type of latch or flip-flop in larger logical circuits, as shown in the two bottom circuits of Figure 7.

That's all for this installment. Coming next month is a look at the brainier side of binary logic. Here is where we put the above logic gates and memory cells to work on mathematical problems. **NV**

Cables • Computer • Communications • Network • Audio • Video

Roger's Systems Specialist Inc.

Mon. - Fri. 8:30am - 5:30pm

800-366-0579
661-295-5577
FAX 661-295-8777

Saturday 9am - 1pm

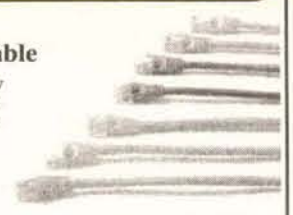
24895 Avenue Rockefeller
Valencia, California 91355

"We Have Great Connections"

www.RogersSystems.com

CAT. 5 CABLE

Also available in many colors!!



Grey

TE-038-L5	3ft. Straight Patch	$1.75
TE-078-L5	7 ft. Straight Patch	$2.00
TE-128-L5	14ft. Straight Patch	$3.00
TE-258-L5	25ft. Straight Patch	$5.00
TE-508-L5	50 ft. Straight Patch	$8.00
TE-758-L5	75ft. Straight Patch	$14.00
TE-108-L5	100 ft. Straight Patch	$16.00

USB

CC-USB-AB6	6ft. USB "A"-"B" M/M	$5.00
CC-USB-AB10	10ft.USB"A"-"B" M/M	$6.00
CC-USB-AB15	15ft.USB "A"-"B" M/M	$8.00
CC-USB-X6	6ft. USB "A"-"A" M/F	$5.00
CC-USB-X10	10ft. USB "A"-"A" M/F	$6.00

USB to Parallel Printer

$23.99

CC-USB-PP

- USB 1.0 specification campatablity Plug & Play and Hot swapable for easy use. Hardware initialized and managed automatic negotiation for fastest protocol.
- Transparently Supporting true bi-directional communications.
- Standard PC parallel port register based operation • Provide multiple logical channels. Enable Microsoft Windows printer driver to print to USB Maximum throughput to 1.216Mbps (ECP compatable)
- No external power supply is required

USB Port to Ethernet 10/100

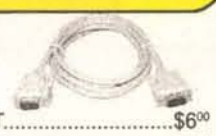
Package Includes:
10/100 Mbps USB Ethernet Adapter
USB Ethernet driver CD
User's manual

TM-USB-110 $69.00

Features:
USB specification 1.0 and 1.1 compliant
Full-Speed USB Device
Supports 1 USB configuration and 1 interface
Supports all USB standard commands
Supports automatically load vender ID from EEPROM

S-VGA Extensions

male/female black



CC-VGA-4C	6FT.	$6.00
CC-VGA-5C	10FT.	$8.00
CC-VGA-25CX	25FT.	$16.00
CC-VGA-50CX	50FT.	$25.00
CC-VGA-100CX	100FT.	$44.00

S-VGA Switch Box Cable

male/male black

CC-VGA-3C	6FT.	$6.00
CC-VGA-9C	10FT.	$8.00
CC-VGA-11C	25FT.	$16.00
CC-VGA50MM	50FT.	$25.00
CC-VGA100MM	100FT.	$44.00

These premium VGA cables are made with 75 ohm coaxial cables. They are triple shielded to support extremely high bandwidth and unsurpassed protection against interference. Furthermore, our premium cables are Plug-N-Play ready and are compatible with the latest technology.

VGA Monitor Cables

Have you just upgraded to a larger monitor, and noticed ghosting images and slow data transmission?
Provides as a replacement video and high-resolution multimedia monitor cable with a HDDB15 connector to your PC.
Recommended for monitors 17" and larger.

CC-VGA-3	6ft.	Male/Male	$6.00
CC-VGA-9	10ft.	Male/Male	$8.00
CC-VGA-4	6ft.	Male/Female	$6.00
CC-VGA-5	10ft.	Male/Female	$8.00

KVM Cable Sets

1 SVGA 2 PS/2

CC-PS2-VGA6	6ft. Male / Male	$12.00
CC-PS2-VGA1	10ft. Male / Male	$16.50
CC-PS2-VGAF	6ft. Male / Female	$14.00
CC-PS2-VGA1F	10ft. Male / Female	$17.00

USB HUB

4-port USB hub with power & cable.
Full compliance w/USB spec. Rev 1.0.
LED indicator for fault or dummy USB port.
Transmission for 5 meter cable segment.
Plug & Play capability for outside peripherals.
Support UHCI and OHCI spec.
One year factory warranty!

$28.95

TM-USB-4HUB

IEEE-1394 FIREWIRE

$10.00

FW-6X4-6	6ft 6pin x 4pin	$10.00
FW-6X4-10	10ft 6pin x 4pin	$12.00
FW-4X4-6	6ft 4pin x 4pin	$10.00
FW-4X4-10	10ft 4pin x 4pin	$12.00

DONGLE

3 COM Dongle for PCMCIA Network Cards.
Designed for 10 BaseT Cards.

$10.00

CA-3COM-TP

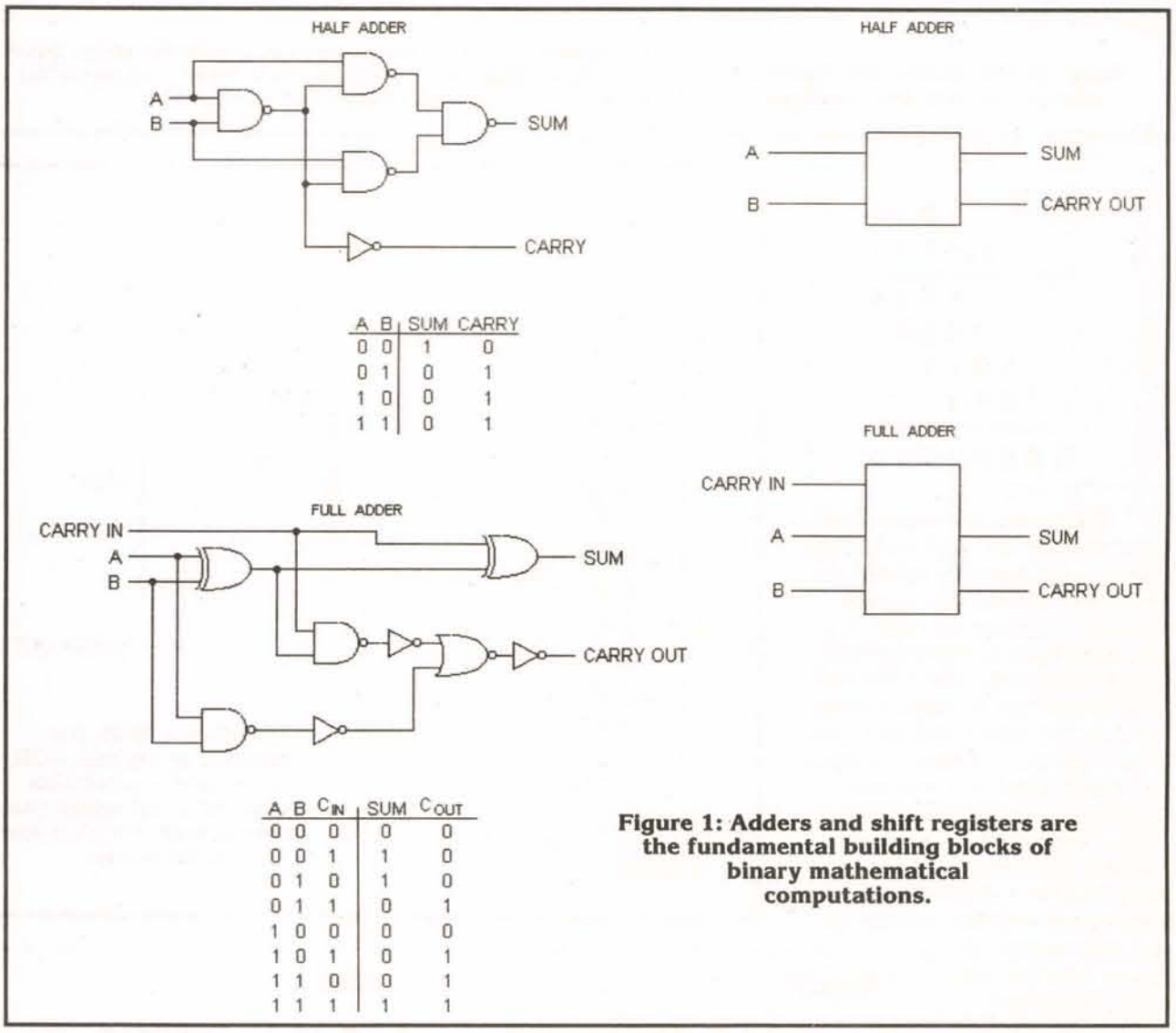
CPU's-Motherboards-HardDrives
Memory -SCSI Adaptors
SCSI Cables - CD burners
CD's & Rewritable CD's
And Much Much More!!!!!

Circle #83 on the Reader Service Card.

Small Logic Gates Spawn Big Dreams Part 2

Binary math for dummies: Discover how binary adders and shift registers can turn simple logic gates into cool calculators.

by TJ Byers



A	B	SUM	CARRY
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

A	B	C_{IN}	SUM	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 1: Adders and shift registers are the fundamental building blocks of binary mathematical computations.

Now that you have a handle on binary logic, and how to make simple gate substitutions to solve your custom IC or obsolete part replacement problems ("Small Logic Gates Spawn Big Dreams — Part 1"), the next step is to put these gates to work for you. You know, the mundane tasks of add, subtract, multiply, and divide. The stuff we learned in elementary school and promptly forgot the moment we pocketed our first calculator.

Unlike "human" math, which is based on the number 10 (a result of having five fingers on each hand), computer math is based on the number 2 — which has the values of 0 and 1. So how do you do math using just nothing and something? The same way it's done using the numbers 0 through 9. The only difference is in the way the 1s and 0s are moved around to fill the needs of borrow and carry.

All binary math operations are built around just two basic circuits: the binary adder and the shift register. While both circuits are made up of several more elementary logic gates, the focus will be on how these two functions perform as a unit. I won't take a microscopic tour of each electron's movement. Instead, I'm going to tell you how to wire the functions together and just what to expect when you flip the switch.

Binary Addition

Basic to all math operations is the binary adder, which comes in two flavors: a half adder and a full adder (Figure 1). The half adder simply tallies two binary bits and outputs a sum. For example:

0 + 0 = 0
1 + 0 = 1
0 + 1 = 1

Nothing surprising here. But what happens when you add 1 + 1? Exactly the same thing that happens when you add 9 + 1 — you get 10. Like decimal addition, binary addition carries over the next most significant digit when the total exceeds the base number. For logic circuits, that's when the sum exceeds 1, whereupon the most-significant digit (MSB) is shifted left one position and a place holder (0) fills the least-most significant (LSB) position. Consequently, the sum of 1 + 1 = 10.

When adding numbers larger than two, a full adder is needed to deal with the overflow, which is called a Carry Out bit. Take the example shown in Table 1 of 1101101 + 101, which has a result of 1110010.

This operation requires an eight-bit adder, which is easily made using a pair of four-bit full adders, like the 74LS83 shown in Figure 2.

Table 1

```
  1 1 1   1      Carry Out
1 1 0 1 1 0 1    A
+       1 0 1    B
---------------
1 1 1 0 0 1 0    Sum
```

Table 2

```
  12          1 0 1 1
x 3           x     1
-----         -------
  36          1 0 1 1
```

Binary Subtraction

Binary subtraction is interest-

Table 3

```
 123          1 0 1 1
x11           x   1 1
-----         -------
 123          1 0 1 1
123           1 0 1 1
-----         -------
1353          1 0 0 0 0 1
```

DesignNotes.com
Your Design Resource on the Web

Improve Your Design Skills, Find Project Advice and More

Velleman DVM890
3-¾ DMM with Capacitance, Frequency and Temperature Measurement (+/-0.5% Accuracy)
Comes complete with probes and holster.
$39.95
For complete specs, visit:
www.designnotes.com

Visit Our Online Forum

On-Line Circuit Archive
Hundreds of Circuits.
Over 23 Different Topics

Designing for Dollars
Submit your favorite circuit or program. Each month the best design entry (Judged by your peers) wins **$100** in cash. Monthly winners are eligible for the yearly **$1200** Grand Prize!

Share What You Know and Learn What You Don't

Visit Us at
www.designnotes.com

Circle #104 on the Reader Service Card.

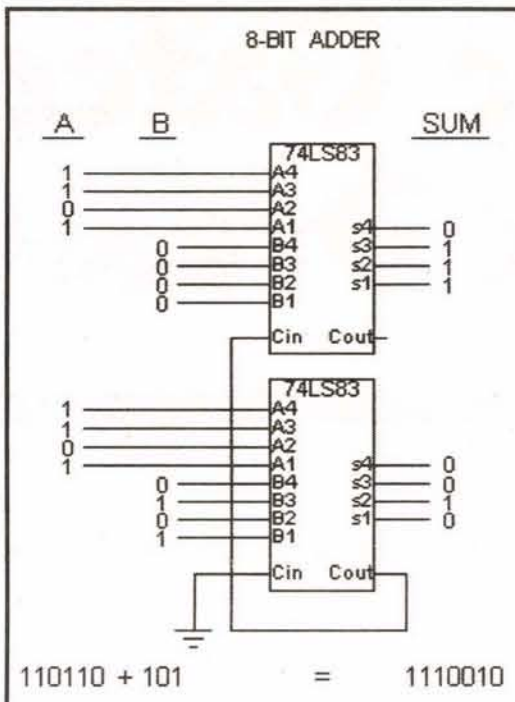
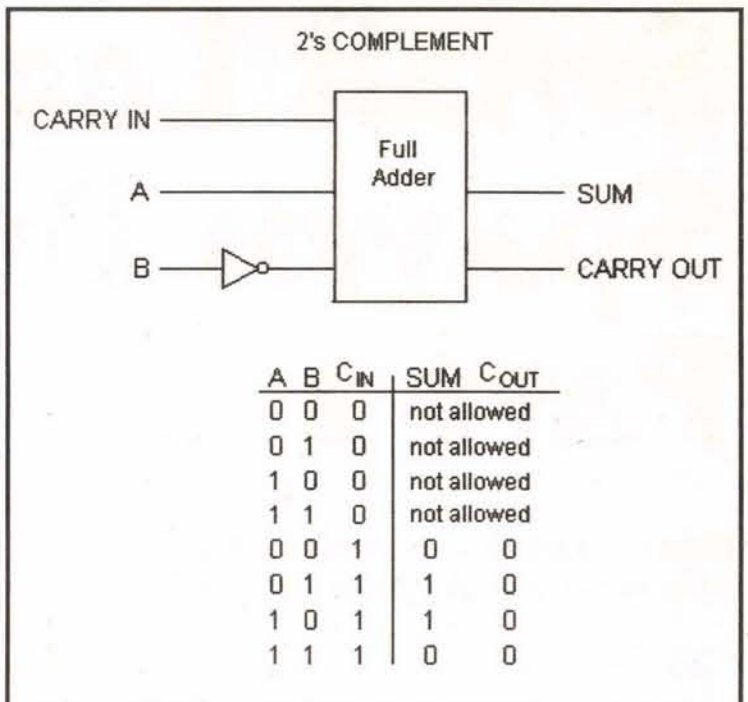


Figure 2: Full adders are stacked to process the required word size.



A	B	C_{IN}	SUM	C_{OUT}
0	0	0	not allowed	
0	1	0	not allowed	
1	0	0	not allowed	
1	1	0	not allowed	
0	0	1	0	0
0	1	1	1	0
1	0	1	1	0
1	1	1	0	0

Figure 3: A 2's complement conversion can be done by adding an inverter in the B input and applying a 1 to Carry In of a full adder.

Table 4

```
    1011
  x 1101
  ------
    1011
   0000
  1011
 1011
 --------
 10001111
```

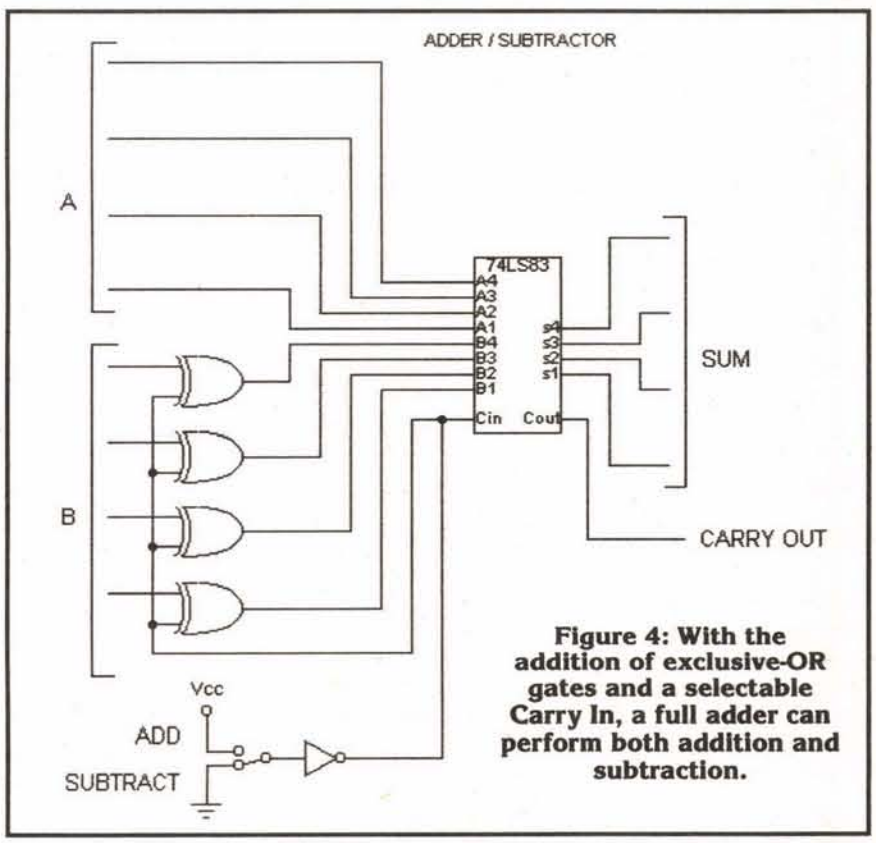


Figure 4: With the addition of exclusive-OR gates and a selectable Carry In, a full adder can perform both addition and subtraction.

ing in that it uses negative numbers to arrive at a result. For example, if you start with 7 and subtract 5, it's the same thing as adding 7 to -5.

7 - 5 = 2

is the same thing as:

7 + (-5) = 2

It's just a different way of skinning a cat, and a concept that wasn't available until the zero was fully understood. In fact, it wasn't until 1657 that a mathematician (John Hudde) used a single variable to represent either a positive or a negative number. For all those years until 1657, positive and negative numbers were handled as separate special cases. The reason is because we couldn't conceive of there being less than nothing.

Computers and logical math are a lot like our ancestors. They don't understand the concept of less than nothing. For a math circuit to perform an operation, it has to have something tangible to work with. That's why subtraction is such an alien concept. In the computer's eyes, you can't have less than nothing — it doesn't exist (which is true; it only exists in our minds and mortgage ledgers). Boolean algebra solves this dilemma by assigning every number a value — even if that value is negative. In essence, you have a stack of apples, let's say, that need to be added and another stack of imaginary (negative) apples to be subtracted. The second stack doesn't exist in reality, they are merely items to be shuffled about. By matching the apples from the positive stack to those of the negative stack — that is, each time a negative apple mates with a positive apple, both are removed from the total — we arrive at an answer.

Still with me? Let's say we have four apples and we need two apples for another project. The computerese way to do this is to give two of the apples a negative value (-2 apples), while leaving the whole (4 apples) a positive value. These two numbers are now entered into a full adder circuit, which spits out the result of 2.

4 apples + (-2) apples = 2 apples

Table 5

```
    1011   Multiplicand
  x 1101   Multiplier
  ------
    1011   First Product
 1011      Shift 2 places
 ---------
  110111   Subtotal
 1011      Next Product
 ---------
10001111   Result
```

Table 6

```
100011   Dividend
  1001   Divisor
 -----
 11010   <> Check; Iter 1
  1001   Divisor
 -----
010001   <> Check; Iter 2
  1001   Divisor
 -----
001000   <> Check; Stop
Result
    11   Divisions
  1000   Remainder
```

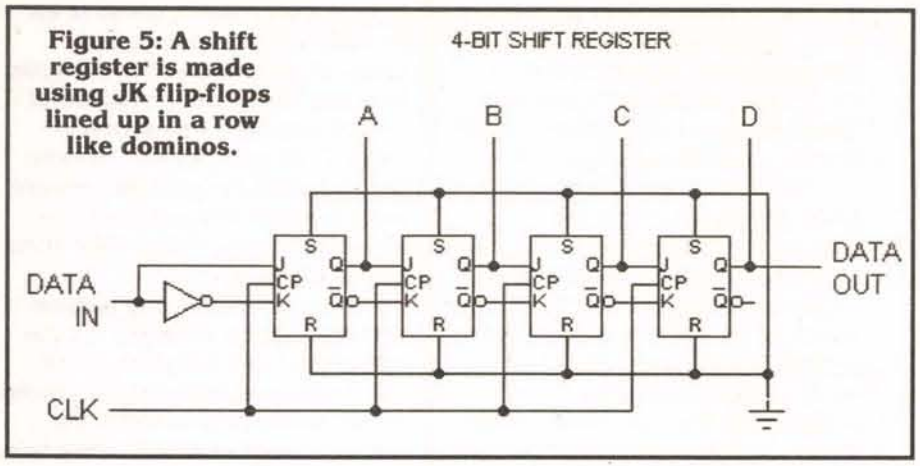


Figure 5: A shift register is made using JK flip-flops lined up in a row like dominos.

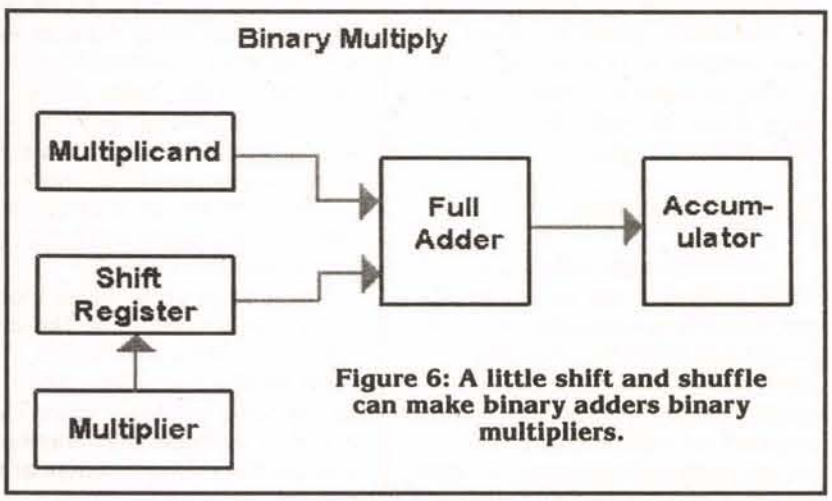


Figure 6: A little shift and shuffle can make binary adders binary multipliers.

Table 8

```
01110  00000  Dividend
       10001  Divisor
       01111  2's Complement
-------------------------------
01110  00000  Dividend
11100  0000   Shift Dividend Left
01111         Add 2's
01011         Result #1
              <>=1; Q=00001
10110         Shift #1 Left
01111         Add 2's
00101         Result #2
              <>=1; Q=00011
01010         Shift #2 Left
01111         Add 2's
11001         Result #3
              <>=0; Q=00110
10001         Add Divisor
01010         Result #4
10100         Shift #4 Left
01111         Add 2's
00011         Result #5
              <>=1; Q=01101
00110         Shift #5 Left
01111         Add 2's
10101         Result #6
              <>=0; Q=11010
10001         Add Divisor
00110         Result #7
              Stop
11010         Quotient
00110         Remainder
```

Simple enough sure, but confusing for a logic gate. Fortunately, there's a binary shortcut that makes the task even easier. It's called 2's complement. If you do a little math here (I'll spare you the details), you'll discover that binary subtraction is identical to adding the A integer to the 2's complement of the B integer. The 2's complement of a number is equal to its NOT (inverted) value plus 1. To find the 2's complement of binary 3, for example, invert 0011 into 1100, then add a 1 (0011 +1) to give 0100 (-3). That's all there is to it. Here's a short list that should give you a grasp of the concept.

Table 7

```
      28  r6
17 | 448
     34
    ---
     108
     102
    ---
       6
```

Binary 2's Complement

Decimal Number	Binary Number	2's Complement (NOT + 1)
0	0000	0001
1	0001	0010
2	0010	0011
3	0011	0100
4	0100	0101
5	0101	0110
6	0110	0111

Why add a 1 to the inversion, you may ask? For the same reason the new Millennium started at 2001 and not 2000. Logic circuits can't deal with the number zero when doing calculations, just like the calendar can't deal with the gap between 1BC and 1AD — that is, there was either a Christ or there wasn't. One AD represents his presence and 1BC is before his birth. There was never a time in-between. Computer logic is the same way. There is never a time when a number in neither positive or negative — it has to be one or the other. Adding a 1 shifts the inverted number back into the realm of computer comprehension.

The 2's complement conversion can be done at the hardware level using an inverter in series with the B input and applying a 1 to the Carry In line of a full adder (Figure 3). This input is then processed by the full adder to arrive at the difference between the two numbers (A and the 2's complement of B).

Since the subtractor just described is based on a full adder, it's possible to construct a multi-purpose adder/subtractor circuit which performs either addition or subtraction, depending on the presence or absence of a 1 on the exclusive-OR/Carry In line. When Carry In is logic 1, the circuit behaves as a subtractor. Pulling Carry In low (logic 0) causes it to perform as an adder. (In our

The RF Connection
213 North Frederick Ave.
Suite 11NV
Gaithersburg, MD USA
20877

http://www.therfc.com/

Complete Selection of MIL-Spec Coax, RF Connectors and Relays

UG-21B/U N Male for RG-213/214.....$5.00
UG-21D/U N Male for RG-213/214.....$3.25

N Connectors for 9913/Flexi4XL/9096

UG-21B/9913.....$6.00 **Pins Only**.....$1.50
UG-21D/9913.....$4.00 **Extra Gasket**..... .75

Amphenol 83-1SP-1050 PL-259.....$0.90
UG-176/U Reducer RG-59/8X. .25 or 5/$1.00
UG-175/U Reducer RG-58/58A .25 or 5/$1.00
Silver Teflon PL-259/Gold Pin.....
.....$1.00 or 10/$9.00

MIL-Spec Coax Available (Teflon, PVC IIA)

New Product: Belden 9913F. 9913 with High Density PE Foam dielectric, stranded center cond. and Duobond III Jacket.....
..... .80/ft or $76.00/100ft

Also New: 9092, RG8X with Type II Jacket
Intro Price.....$23.00/100ft

Call for Specials of the Month

Full Line of Audio Connectors for Icom, Kenwood, and Yaesu

8 Pin Mike Female.....$2.50
8 Pin Mike Male Panel.....$2.50
13 Pin DIN for Kenwood.....$2.75
8 Pin DIN for Icom.....$1.00
8 Pin DIN for Kenwood.....$1.50

Prices Do Not Include Shipping

Orders 800/783-2666
Info 301/840-5477
FAX 301/869-3680

Circle #103 on the Reader Service Card.

PICmicros & BASIC

PicBasic Compiler - $99.95
PicBasic Pro Compiler - $249.95

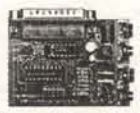
Now it's even easier to program the fast and powerful Microchip PICmicros. The PicBasic and PicBasic Pro Compilers convert your English-like BASIC programs to files that can be put directly into a PICmicro. True compilers for faster, longer programs. BASIC Stamp™ I/II* libraries. For mid-range PIC12C67x, 14Cxxx, 16C55x, 6xx, 7xx, 8x, 87x, 9xx and high-end 17Cxxx (PicBasic Pro only).

*BASIC Stamp is a registered trademark of Parallax Inc.

New! PIC-X1 Experimenter/ Lab Board

Assembled - $199.95
Kit with parts - $139.95
Bare PCB only - $49.95

EPIC Plus PIC Programmer - $59.95



Programs PIC12C5xx, 67x, 14Cxxx, 16C505, 55x, 6xx, 7xx, 8x, 87x and 9xx. Optional ZIF adapters for DIP, SOIC, MQFP, PLCC. Runs off two 9-volt batts or optional AC adapter. Includes programming software and assembler.

PICProto Prototyping Boards

Get it wired quicker! High-quality blank prototyping boards for PICmicros. Holds PICmicro, 5V reg, caps, oscillator, DB9-25, large proto area. $8.95 - $19.95

microEngineering Labs, Inc.
Box 7532 Colorado Springs CO 80933
(719) 520-5323 (719) 520-1867 fax
http://www.melabs.com

Circle #105 on the Reader Service Card

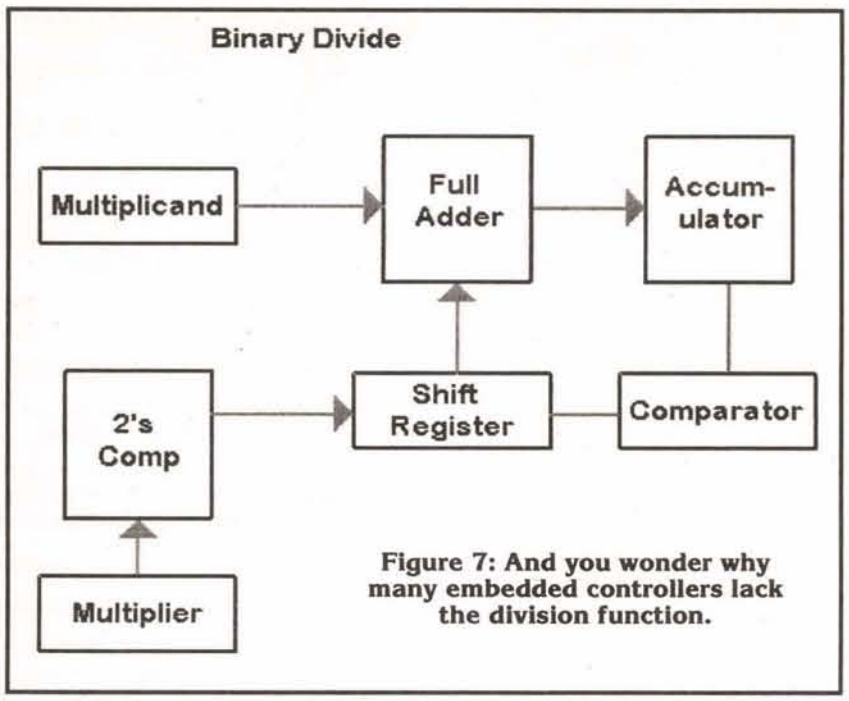


Figure 7: And you wonder why many embedded controllers lack the division function.

example of Figure 4, please note that the ADD/SUBTRACT switch goes through an inverter; don't let this NOT hardware gate confuse you when following the above discussion.)

Binary Multiplication

Any school kid knows that multiplication is simply a series of additions done a specified number of times. For example, 3 x 4 is the same as adding 3 four times over, or 3 + 3 + 3 + 3. With binary multiplication we do the same thing — add up a number the required number of times and arrive at a result.

We also learned very early that there is simple multiplication, where one number is multiplied by a single digit, and compound multiplication, where numbers of two digits or more are multiplied together. Simple multiplication looks like what you see in Table 2, whereas compound multiplication looks like what you see in Table 3.

Notice the shift and add technique which is the signature pattern of compound multiplication. Also notice that it's used with both decimal and binary multiplication. Shifting the position of the line one space to the left is equivalent to multiplying by 2 (binary) or by 10 (decimal). Here's where the shift register, mentioned earlier, comes into play. A shift register is made using JK flip-flops all lined up in a row like dominos, as shown in Figure 5. Let's look at a typical binary multiplication and see where it takes us at. Check Table 4. This is a straightforward calculation using the rules we learned in PS3. The shift register is first loaded with the multiplicand. Then the number in the register is multiplied by the multiplier. After the first line is completed, the register shifts its contents one position to the left and the process is repeated. This continues until all digits of the multiplier are exhausted. Just like doing compound multiplication using decimal numbers, and this circuitry is simple and straightforward, as shown in Figure 6.

But if you look at it carefully, you'll see places where the process can be streamlined to better fit the computer community. To begin with, 0 times anything is always 0, so line 2 is superfluous and can be removed. And it's more efficient and faster to do subtotals as lines are added rather than crunch through a tall stack of numbers at the end, which results in the pattern shown in Table 5. As the numbers grow larger, this winnowing process becomes increasingly more efficient, saving both time and resources.

Binary Division

Of all the arithmetic operations, division is the most complicated and can consume the most resources. In many computer applications, division is less frequently used than addition, subtraction, or multiplication. As a result, some microprocessors that are designed for digital signal processing (DSP) or embedded processor applications do not have a divide instruction. When hardware division is needed, there are several methods to choose from, none of which are stand-outs.

The simplest is to start subtracting the divisor from the dividend until you run out of numbers to take away from. With this scheme, all you have to do is count the number of iterations — and that's it. For example, look at Table 6 to see how 35 divided by 9 looks in binary using this method. Or, you can do it the old-fashioned way using long division. That is, look at the problem, guess at an answer, test your guess, and respond accordingly. For example, let's take the decimal problem of 448 divided by 17 as shown in Table 7. As you can see, this can be a long-winded process because each iteration takes at least one tick of the clock. Fortunately, there are other options.

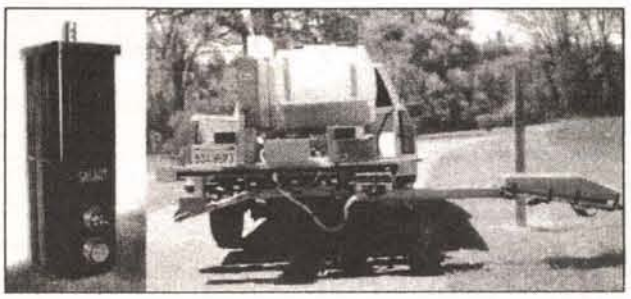
We can also do long division with logic gates. Basically, it's the opposite of multiplication, where the dividend is shifted right, instead of left, and subtracted from the divisor instead of added. While this seems great in theory, it can get rather cumbersome. A better way is to use 2's complements, and proceed just like before using straightforward multiplication — but with a twist. After each subtraction, a comparison has to be made to determine if the result of the last calculation is larger or smaller than the divisor. See Table 8.

Yes, the process looks more complicated — and it is! The circuitry, too, is more complex because of the compare and decide logic needed (Figure 7). But it's a lot faster than waiting for a count-down loop to arrive at an answer.

And The Numbers Are In

Now that you've taken the whole binary tour, I hope that I've touched on something that affects your electronics hobby needs, whether it be scarce part substitutions or basic design. Or, at least piqued an interest or exposed you to something new. Nonetheless, you have to agree that it's so amazing that even the most complex of digital designs are based on three lowly Boolean functions: AND, OR, and NOT. **NV**

Taking the Backache Out of Roadside Weed Spraying



Noxious weeds growing in the cracks and under the guard rails along the nation's roadways cause millions of dollars in damage every year. The imposed challenge is to find an efficient, cost-effective, and environmentally-safe solution that will eliminate the weeds. The SmartMotor™, developed by Animatics Corporation in Santa Clara, CA, helped to solve this problem by employing a unique approach.

The SmartMotor is a stand-alone, brushless DC servomotor with built-in controller, drive, and network capability. A NEMA 56 size SmartMotor is used to actuate a boom in a roadside weed spraying system developed by Microphor in Willits, CA. The automated system mounts the motor, a 110 gallon spray tank, and other hardware on the bed of a pick-up truck. Weed killer is sprayed from both a fixed boom underneath the truck, as well as a retractable boom that rotates and extends five feet to each side of the vehicle to cover a range of 13 feet. As the truck drives along dirt or paved roads, a light sensing spray head activates when the light spectrum associated with chlorophyll is detected. A total of 13 spray heads independently seek and spray individual weeds, but do not spray bare surfaces.

To spray weeds and avoid obstacles while maintaining a truck speed of between 5-15 mph, the motion of the boom must be affected quickly. The high torque of the SmartMotor is enough to extend and retract the boom in half a second. Sensors at the front of the truck register obstacles such as utility poles and street signs. The spray arm automatically retracts when an obstacle larger than ¼" x 2" is within sensor range. The SmartMotor receives inputs from the truck's radar, speedometer, and optical sensors.

As the SmartMotor scans the various inputs, the controller constantly makes real-time calculations of acceleration and velocity needed to position the boom. A motion control program is stored on a memory module in the motor itself. The program employs a very rich command set that includes motion control, arithmetic functions, Boolean operators, and conditional logic.

The resultant system is a self-contained, mobile, highly-efficient package that yields the operator significant savings in chemical costs. Compared to conventional systems, 50% to 80% of chemicals costs savings are attained. Additionally, reducing the amount of applied weed killer addresses growing concerns about the overuse of agricultural chemicals.

For more information, go to www.animatics.com or contact the Animatics office at 408-748-8721.